PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 7/50, 7/52, 15/16, 15/80, 17/10, 17/14 — A1

(11) International Publication Number: **WO 00/22503**

(43) International Publication Date: 20 April 2000 (20.04.00)

(21) International Application Number: PCT/US99/23494

(22) International Filing Date: 8 October 1999 (08.10.99)

(30) Priority Data:

60/103,712	9 October 1998 (09.10.98)	US
09/337,839	22 June 1999 (22.06.99)	US

(71) Applicant: BOPS INCORPORATED [US/US]; Suite 210, 6340 Quadrangle Drive, Chapel Hill, NC 27514 (US).

(72) Inventors: PITSIANIS, Nikos, P.; Apartment 11, 6205 Farrington Road, Chapel Hill, NC 27514 (US). PECHANEK, Gerald, G.; 107 Stoneleigh Drive, Cary, NC 27511 (US). RODRIGUEZ, Ricardo, E., 9116 Brinsbrook Drive, Raleigh, NC 27613 (US).

(74) Agent: PRIEST, Peter, H.; Law Offices of Peter H. Priest, 529 Dogwood Drive, Chapel Hill, NC 27516 (US).

(81) Designated States: CA, CN, IL, JP, KR, MX, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published

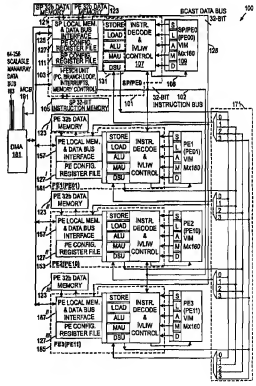
With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: EFFICIENT COMPLEX MULTIPLICATION AND FAST FOURIER TRANSFORM (FFT) IMPLEMENTATION ON THE MANARRAY ARCHITECTURE

(57) Abstract

Efficient computation of complex multiplication results and very efficient fast Fourier transforms (FFTs) are provided. A parallel array VLIW digital signal processor (100) is employed along with specialized complex multiplication instructions and communication operations between the processing elements (101, 151, 153, 155) which are overlapped with computation to provide very high performance operation. Successive iterations of a loop of tightly packed VLIWs (100) are used allowing the complex multiplication pipeline hardware to be efficiently used. In addition, efficient techniques for supporting combined multiply accumulate operations are described.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

Efficient Complex Multiplication and Fast Fourier Transform (FFT) Implementation on the ManArray Architecture

This application claims the benefit of U.S. Provisional Application Serial No. 60/103,712 filed October 9, 1998 which is incorporated by reference in its entirety herein.

Field of the Invention

The present invention relates generally to improvements to parallel processing, and more particularly to methods and apparatus for efficiently calculating the result of a complex multiplication. Further, the present invention relates to the use of this approach in a very efficient FFT implementation on the manifold array ("ManArray") processing architecture.

Background of the Invention

The product of two complex numbers x and y is defined to be $z = x_R y_R - x_I y_I + i(x_R y_I + x_I y_R)$, where $x = x_R + ix_I$, $y = y_R + iy_I$ and i is an imaginary number, or the square root of negative one, with $i^2 = -1$. This complex multiplication of x and y is calculated in a variety of contexts, and it has been recognized that it will be highly advantageous to perform this calculation faster and more efficiently.

Summary of the Invention

The present invention defines hardware instructions to calculate the product of two complex numbers encoded as a pair of two fixed-point numbers of 16 bits each in two cycles with single cycle pipeline throughput efficiency. The present invention also defines extending a series of multiply complex instructions with an accumulate operation. These special instructions are then used to calculate the FFT of a vector of numbers efficiently.

A more complete understanding of the present invention, as well as other features and advantages of the invention will be apparent from the following Detailed Description and the accompanying drawings.

Brief Description of the Drawings

Fig. 1 illustrates an exemplary 2x2 ManArray iVLIW processor;

Fig. 2A illustrates a presently preferred multiply complex instruction, MPYCX;

Fig. 2B illustrates the syntax and operation of the MPYCX instruction of Fig. 2A;

Fig. 3A illustrates a presently preferred multiply complex divide by 2 instruction, MPYCXD2;

Fig. 3B illustrates the syntax and operation of the MPYCXD2 instruction of Fig. 3A;

Fig. 4A illustrates a presently preferred multiply complex conjugate instruction, MPYCXJ;

Fig. 4B illustrates the syntax and operation of the MPYCXJ instruction of Fig. 4A;

Fig. 5A illustrates a presently preferred multiply complex conjugate divide by two instruction, MPYCXJD2;

Fig. 5B illustrates the syntax and operation of the MPYCXJD2 instruction of Fig. 5A;

Fig. 6 illustrates hardware aspects of a pipelined multiply complex and its divide by two instruction variant;

Fig. 7 illustrates hardware aspects of a pipelined multiply complex conjugate, and its divide by two instruction variant;

Fig. 8 shows an FFT signal flow graph;

Fig. 9A-9H illustrate aspects of the implementation of a distributed FFT algorithm on a 2x2 ManArray processor using a VLIW algorithm with MPYCX instructions in a cycle-by-cycle sequence with each step corresponding to operations in the FFT signal flow graph;

Fig. 9I illustrates how multiple iterations may be tightly packed in accordance with the present invention for a distributed FFT of length four;

Fig. 9J illustrates how multiple iterations may be tightly packed in accordance with the present invention for a distributed FFT of length two;

Figs. 10A and 10B illustrate Kronecker Product examples for use in reference to the mathematical presentation of the presently preferred distributed FFT algorithm;

Fig. 11A illustrates a presently preferred multiply accumulate instruction, MPYA;

Fig. 11B illustrates the syntax and operation of the MPYA instruction of Fig. 11A;

Fig. 12A illustratcs a presently preferred sum of 2 products accumulate instruction, SUM2PA;

Fig. 12B illustrates the syntax and operation of the SUM2PA instruction of Fig. 12A;

Fig. 13A illustrates a presently preferred multiply complex accumulate instruction, MPYCXA;

Fig. 13B illustrates the syntax and operation of the MPYCXA instruction of Fig. 13A;

Fig. 14A illustrates a presently preferred multiply complex accumulate divide by two instruction, MPYCXAD2;

Fig. 14B illustrates the syntax and operation of the MPYCXAD2 instruction of Fig. 14A;

Fig. 15A illustrates a presently preferred multiply complex conjugate accumulate instruction, MPYCXJA;

Fig. 15B illustrates the syntax and operation of the MPYCXJA instruction of Fig. 15A;

Fig. 16A illustrates a presently preferred multiply complex conjugate accumulate divide by two instruction, MPYCXJAD2;

Fig. 16B illustrates the syntax and operation of the MPYCXJAD2 instruction of Fig. 16A;

Fig. 17 illustrates hardware aspects of a pipelined multiply complex accumulate and its divide by two variant; and

Fig. 18 illustrates hardware aspects of a pipelined multiply complex conjugate accumulate and its divide by two variant.

Detailed Description

Further details of a presently preferred ManArray architecture for use in conjunction with the present invention are found in U.S. Patent Application Serial No. 08/885,310 filed June 30, 1997, U.S. Patent Application Serial No. 08/949,122 filed October 10, 1997, U.S. Patent Application Serial No. 09/169,255 filed October 9, 1998, U.S. Patent Application Serial No. 09/169,256 filed October 9, 1998, U.S. Patent Application Serial No. 09/169,072 filed October 9, 1998, U.S. Patent Application Serial No. 09/187,539 filed November 6, 1998, U.S. Patent Application Serial No. 09/205,558 filed December 4, 1998, U.S. Patent Application Serial No. 09/215,081 filed December 18, 1998, U.S. Patent Application Serial No. 09/228,374 filed January 12, 1999, U.S. Patent Application Serial No. 09/238,446 filed January 28, 1999, U.S. Patent Application Serial No. 09/267,570 filed March 12, 1999, as well as, Provisional Application Serial No. 60/092,130 entitled "Methods and Apparatus for Instruction Addressing in Indirect VLIW Processors" filed July 9, 1998, Provisional Application Serial No. 60/103,712 entitled "Efficient Complex Multiplication and Fast Fourier Transform (FFT) Implementation on the ManArray" filed October 9, 1998, Provisional Application Serial No. 60/106,867 entitled "Methods and Apparatus for Improved Motion Estimation for Video Encoding" filed November 3, 1998, Provisional Application Serial No. 60/113,637 entitled "Methods and Apparatus for Providing Direct Memory Access (DMA) Engine" filed December 23, 1998 and Provisional Application Serial No. 60/113,555 entitled "Methods and Apparatus Providing Transfer Control" filed December 23, 1998, respectively, and incorporated by reference herein in their entirety.

In a presently preferred embodiment of the present invention, a ManArray 2x2 iVLIW single instruction multiple data stream (SIMD) processor 100 shown in Fig. 1 contains a controller sequence processor (SP) combined with processing element-0 (PE0) SP/PE0 101, as described in further detail in U.S. Application Serial No. 09/169,072 entitled "Methods and Apparatus for Dynamically Merging an Array Controller with an Array Processing Element". Three additional PEs 151, 153, and 155 are also utilized to demonstrate the implementation of efficient complex multiplication and fast fourier transform (FFT) computations on the ManArray architecture in accordance with the present invention. It is noted that the PEs can be also labeled with their matrix positions as shown in parentheses for PE0 (PE00) 101, PE1 (PE01)151, PE2 (PE10) 153, and PE3 (PE11) 155.

The SP/PE0 101 contains a fetch controller 103 to allow the fetching of short instruction words (SIWs) from a 32-bit instruction memory 105. The fetch controller 103 provides the typical functions needed in a programmable processor such as a program counter (PC), branch capability, digital signal processing, EP loop operations, support for interrupts, and also provides the instruction memory management control which could include an instruction cache if needed by an application. In addition, the SIW I-Fetch controller 103 dispatches 32-bit SIWs to the other PEs in the system by means of a 32-bit instruction bus 102.

In this exemplary system, common elements are used throughout to simplify the explanation, though actual implementations are not so limited. For example, the execution units 131 in the combined SP/PE0 101 can be separated into a set of execution units optimized for the control function, e.g. fixed point execution units, and the PE0 as well as the other PEs 151, 153 and 155 can be optimized for a floating point application. For the purposes of this description, it is assumed that the execution units 131 are of the same type in the SP/PE0 and the other PEs. In a similar manner, SP/PE0 and the other PEs use a five instruction slot iVLIW architecture which contains a very long instruction word memory (VIM) memory 109 and an instruction decode and VIM controller function unit 107 which receives instructions as dispatched from the SP/PE0's I-Fetch unit 103 and generates the VIM addresses-and-control signals 108 required to access the iVLIWs stored in the VIM. These iVLIWs are identified by the letters SLAMD in VIM 109. The loading of the iVLIWs is described in further detail in U.S. Patent Application Serial No. 09/187,539 entitled "Methods and Apparatus for Efficient Synchronous MIMD Operations with iVLIW PE-to-PE Communication". Also contained in the SP/PE0 and the other PEs is a common PE

configurable register file 127 which is described in further detail in U.S. Patent Application Serial No. 09/169,255 entitled "Methods and Apparatus for Dynamic Instruction Controlled Reconfiguration Register File with Extended Precision".

Due to the combined nature of the SP/PE0, the data memory interface controller 125 must handle the data processing needs of both the SP controller, with SP data in memory 121, and PE0, with PE0 data in memory 123. The SP/PE0 controller 125 also is the source of the data that is sent over the 32-bit broadcast data bus 126. The other PEs 151, 153, and 155 contain common physical data memory units 123', 123", and 123"' though the data stored in them is generally different as required by the local processing done on each PE. The interface to these PE data memories is also a common design in PEs 1, 2, and 3 and indicated by PE local memory and data bus interface logic 157, 157' and 157". Interconnecting the PEs for data transfer communications is the cluster switch 171 more completely described in U.S. Patent Application Serial No. 08/885,310 entitled "Manifold Array Processor", U.S. Application Serial No. 09/949,122 entitled "Methods and Apparatus for Manifold Array Processing", and U.S. Application Serial No. 09/169,256 entitled "Methods and Apparatus for ManArray PE-to-PE Switch Control". The interface to a host processor, other peripheral devices, and/or external memory can be done in many ways. The primary mechanism shown for completeness is contained in a direct memory access (DMA) control unit 181 that provides a scalable ManArray data bus 183 that connects to devices and interface units external to the ManArray core. The DMA control unit 181 provides the data flow and bus arbitration mechanisms needed for these external devices to interface to the ManArray core memories via the multiplexed bus interface represented by line 185. A high level view of a ManArray Control Bus (MCB) 191 is also shown.

All of the above noted patents are assigned to the assignee of the present invention and incorporated herein by reference in their entirety.

Special Instructions for Complex Multiply

Turning now to specific details of the ManArray processor as adapted by the present invention, the present invention defines the following special hardware instructions that execute in each multiply accumulate unit (MAU), one of the execution units 131 of Fig. 1 and in each PE, to handle the multiplication of complex numbers:

- MPYCX instruction 200 (Fig. 2A), for multiplication of complex numbers, where the complex product of two source operands is rounded according to the rounding mode specified in the instruction and loaded into the target register. The complex numbers

are organized in the source register such that halfword H1 contains the real component and halfword H0 contains the imaginary component. The MPYCX instruction format is shown in Fig. 2A. The syntax and operation description 210 is shown in Fig. 2B.

- MPYCXD2 instruction 300 (Fig. 3A), for multiplication of complex numbers, with the results divided by 2, Fig. 3, where the complex product of two source operands is divided by two, rounded according to the rounding mode specified in the instruction, and loaded into the target register. The complex numbers are organized in the source register such that halfword H1 contains the real component and halfword H0 contains the imaginary component. The MPYCXD2 instruction format is shown in Fig. 3A. The syntax and operation description 310 is shown in Fig. 3B.
- MPYCXJ instruction 400 (Fig. 4A), for multiplication of complex numbers where the second argument is conjugated, where the complex product of the first source operand times the conjugate of the second source operand, is rounded according to the rounding mode specified in the instruction and loaded into the target register. The complex numbers are organized in the source register such that halfword H1 contains the real component and halfword H0 contains the imaginary component. The MPYCXJ instruction format is shown in Fig. 4A. The syntax and operation description 410 is shown in Fig. 4B.
- MPYCXJD2 instruction 500 (Fig. 5A), for multiplication of complex numbers where the second argument is conjugated, with the results divided by 2, where the complex product of the first source operand times the conjugate of the second operand, is divided by two, rounded according to the rounding mode specified in the instruction and loaded into the target register. The complex numbers are organized in the source register such that halfword H1 contains the real component and halfword H0 contains the imaginary component. The MPYCXJD2 instruction format is shown in Fig. 5A. The syntax and operation description 510 is shown in Fig. 5B.

All of the above instructions 200, 300, 400 and 500 complete in 2 cycles and are pipeline-able. That is, another operation can start executing on the execution unit after the first cycle. All complex multiplication instructions return a word containing the real and imaginary part of the complex product in half words H1 and H0 respectively.

To preserve maximum accuracy, and provide flexibility to programmers, four possible rounding modes are defined:

- Round toward the nearest integer (referred to as ROUND)
- Round toward 0 (truncate or fix, referred to as TRUNC)
- Round toward infinity (round up or ceiling, the smallest integer greater than or equal to the argument, referred to as CEIL)
- Round toward negative infinity (round down or floor, the largest integer smaller than or equal to the argument, referred to as FLOOR).

Hardware suitable for implementing the multiply complex instructions is shown in Fig. 6 and Fig. 7. These figures illustrate a high level view of the hardware apparatus 600 and 700 appropriate for implementing the functions of these instructions. This hardware capability may be advantageously embedded in the ManArray multiply accumulate unit (MAU), one of the execution units 131 of Fig. 1 and in each PE, along with other hardware capability supporting other MAU instructions. As a pipelined operation, the first execute cycle begins with a read of the source register operands from the compute register file (CRF) shown as registers 603 and 605 in Fig. 6 and as registers 111, 127, 127', 127", and 127"' in Fig. 1. These register values are input to the MAU logic after some operand access delay in halfword data paths as indicated to the appropriate multiplication units 607, 609, 611, and 613 of Fig. 6. The outputs of the multiplication operation units, X_R*Y_R 607, X_R*Y_I 609, X_I*Y_R 611, and X_I*Y_I 613, are stored in pipeline registers 615, 617, 619, and 621, respectively. The second execute cycle, which can occur while a new multiply complex instruction is using the first cycle execute facilities, begins with using the stored pipeline register values, in pipeline register 615, 617, 619, and 621, and appropriately adding in adder 625 and subtracting in subtractor 623 as shown in Fig. 6. The add function and subtract function are selectively controlled functions allowing either addition or subtraction operations as specified by the instruction. The values generated by the apparatus 600 shown in Fig. 6 contain a maximum precision of calculation which exceeds 16-bits. Consequently, the appropriate bits must be selected and rounded as indicated in the instruction before storing the final results. The selection of the bits and rounding occurs in selection and rounder circuit 627. The two 16-bit rounded results are then stored in the appropriate halfword position of the target register 629 which is located in the compute register file (CRF). The divide by two variant of the multiply complex instruction 300 selects a different set of bits as

specified in the instruction through block 627. The hardware 627 shifts each data value right by an additional 1-bit and loads two divided-by-2 rounded and shifted values into each half word position in the target registers 629 in the CRF.

The hardware 700 for the multiply complex conjugate instruction 400 is shown in Fig. 7. The main difference between multiply complex and multiply complex conjugate is in adder 723 and subtractor 725 which swap the addition and subtraction operation as compared with Fig. 6. The results from adder 723 and subtractor 725 still need to be selected and rounded in selection and rounder circuit 727 and the final rounded results stored in the target register 729 in the CRF. The divide by two variant of the multiply complex conjugate instruction 500 selects a different set of bits as specified in the instruction through selection and rounder circuit 727. The hardware of circuit 727 shifts each data value right by an additional 1-bit and loads two divided-by-2 rounded and shifted values into each half word position in the target registers 729 in the CRF.

The FFT Algorithm

The power of indirect VLIW parallelism using the complex multiplication instructions is demonstrated with the following fast Fourier transform (FFT) example. The algorithm of this example is based upon the sparse factorization of a discrete Fourier transform (DFT) matrix. Kronecker-product mathematics is used to demonstrate how a scalable algorithm is created.

The Kronecker product provides a means to express parallelism using mathematical notation. It is known that there is a direct mapping between different tensor product forms and some important architectural features of processors. For example, tensor matrices can be created in parallel form and in vector form. J. Granata, M. Conner, R. Tolimieri, The Tensor Product: A Mathematical Programming Language for FFTs and other Fast DSP Operations, *IEEE SP Magazine,* January 1992, pp. 40 - 48. The Kronecker product of two matrices is a block matrix with blocks that are copies of the second argument multiplied by the corresponding element of the first argument. Details of an exemplary calculation of matrix vector products

$$y = (I_m \otimes A)x$$

are shown in Fig. 10A. The matrix is block diagonal with m copies of A. If vector x was distributed block-wise in m processors, the operation can be done in parallel without any

communication between the processors. On the other hand, the following calculation, shown in detail in Fig. 10B,

$$y = (A \otimes \mathrm{Im})x$$

requires that x be distributed physically on m processors for vector parallel computation.

The two Kronecker products are related via the identity

$$I \otimes A = P(A \otimes I)P^T$$

where P is a special permutation matrix called stride permutation and P^T is the transpose permutation matrix. The stride permutation defines the required data distribution for a parallel operation, or the communication pattern needed to transform block distribution to cyclic and vice-versa.

The mathematical description of parallelism and data distributions makes it possible to conceptualize parallel programs, and to manipulate them using linear algebra identities and thus better map them onto target parallel architectures. In addition, Kronecker product notation arises in many different areas of science and engineering. The Kronecker product simplifies the expression of many fast algorithms. For example, different FFT algorithms correspond to different sparse matrix factorizations of the Discrete Fourier Transform (DFT), whose factors involve Kronecker products. Charles F. Van Loan, *Computational Frameworks for the Fast Fourier Transform*, SIAM, 1992, pp 78-80.

The following equation shows a Kronecker product expression of the FFT algorithm, based on the Kronecker product factorization of the DFT matrix,

$$F_n = (F_p \otimes I_m)D_{p,m}(I_p \otimes F_m)P_{n,p}$$

where:

n is the length of the transform

p is the number of PEs

m = n/p

The equation is operated on from right to left with the $P_{n,p}$ permutation operation occurring first. The permutation directly maps to a direct memory access (DMA) operation that specifies how the data is to be loaded in the PEs based upon the number of PEs p and length of the transform n.

$$F_n = (F_p \otimes I_m)D_{p,m}(I_p \otimes F_m)P_{n,p}$$

where $P_{n,p}$ corresponds to DMA loading data with stride p to local PE memories.

In the next stage of operation all the PEs execute a local FFT of length m=n/p with local data. No communications between PEs is required.

$$F_n = (F_p \otimes I_m)D_{p,m}(I_p \otimes F_m)P_{n,p}$$

where $(I_p \otimes F_m)$ specifies that all PEs execute a local FFT of length m sequentially, with local data.

In the next stage, all the PEs scale their local data by the twiddle factors and collectively execute m distributed FFTs of length p. This stage requires inter-PE communications.

$$F_n = (F_p \otimes I_m)D_{p,m}(I_p \otimes F_m)P_{n,p}$$

where $(F_p \otimes I_m)D_{p,m}$ specifies that all PEs scale their local data by the twiddle factors and collectively execute multiple FFTs of length p on distributed data. In this final stage of the FFT computation, a relatively large number m of small distributed FFTs of size p must be calculated efficiently. The challenge is to completely overlap the necessary communications with the relatively simple computational requirements of the FFT.

The sequence of illustrations of Figs. 9A-9H outlines the ManArray distributed FFT algorithm using the indirect VLIW architecture, the multiply complex instructions, and operating on the 2x2 ManArray processor 100 of Fig. 1. The signal flow graph for the small FFT is shown in Fig. 8 and also shown in the right-hand-side of Figs. 9A-9H. In Fig. 8, the operation for a 4 point FFT is shown where each PE executes the operations shown on a horizontal row. The operations occur in parallel on each vertical time slice of operations as shown in the signal flow graph figures in Figs. 9A-9H. The VLIW code is displayed in a tabular form in Figs 9A-9H that corresponds to the structure of the ManArray architecture and the iVLIW instruction. The columns of the table correspond to the execution units available in the ManArray PE: Load Unit, Arithmetic Logic Unit (ALU), Multiply Accumulate Unit (MAU), Data Select Unit (DSU) and the Store Unit. The rows of the table can be interpreted as time steps representing the execution of different iVLIW lines.

The technique shown is a software pipeline implemented approach with iVLIWs. In Figs. 9A-9I, the tables show the basic pipeline for PE3 155. Fig. 9A represents the input of the data X and its corresponding twiddle factor W by loading them from the PEs local

memories, using the load indirect (Lii) instruction. Fig. 9B illustrates the complex arguments X and W which are multiplied using the MPYCX instruction 200, and Fig. 9C illustrates the communications operation between PEs, using a processing element exchange (PEXCHG) instruction. Further details of this instruction are found in U.S. Application Serial No. 09/169,256 entitled "Methods and Apparatus for ManArray PE-PE Switch Control" filed October 9, 1998. Fig. 9D illustrates the local and received quantities are added or subtracted (depending upon the processing element, where for PE3 a subtract (sub) instrution is used). Fig. 9E illlustrates the result being multiplied by -i on PE3, using the MPYCX instruction. Fig. 9F illustrates another PE-to-PE communications operation where the previous product is exchanged between the PEs, using the PEXCHG instruction. Fig. 9G illustrates the local and received quantities are added or subtracted (depending upon the processing element, where for PE3 a subtract (sub) instruction is used). Fig. 9H illustrates the step where the results are stored to local memory, using a store indirect (sii) instruction.

The code for PEs 0, 1, and 2 is very similar, the two subtractions in the arithmetic logic unit in steps 9D and 9G are substituted by additions or subtractions in the other PEs as required by the algorithm displayed in the signal flow graphs. To achieve that capability and the distinct MPYCX operation in Fig. 9E shown in these figures, synchronous MIMD capability is required as described in greater detail in United States Patent Application Serial No. 09/187,539 filed November 6, 1998 and incorporated by reference herein in its entirety. By appropriate packing, a very tight software pipeline can be achieved as shown in Fig. 9I for this FFT example using only two VLIWs.

In the steady state, as can be seen in Fig. 9I, the Load, ALU, MAU, and DSU units are fully utilized in the two VLIWs while the store unit is used half of the time. This high utilization rate using two VLIWs leads to very high performance. For example, a 256-point complex FFT can be accomplished in 425 cycles on a 2x2 ManArray.

As can be seen in the above example, this implementation accomplishes the following:

- An FFT butterfly of length 4 can be calculated and stored every two cycles, using four PEs.
- The communication requirement of the FFT is completely overlapped by the computational requirements of this algorithm.

- The communication is along the hypercube connections that are available as a subset of the connections available in the ManArray interconnection network.
- The steady state of this algorithm consists of only two VLIW lines (the source code is two VLIW lines long).
- All execution units except the Store unit are utilized all the time, which lead us to conclude that this implementation is optimal for this architecture.

Problem Size Discussion

The equation:

$$F_n = (F_p \otimes I_m) D_{p,m} (I_p \otimes F_m) P_{n,p}$$

where:

n is the length of the transform,

p is the number of PEs, and

m = n/p

is parameterized by the length of the transform n and the number of PEs, where m=n/p relates to the size of local memory needed by the PEs. For a given power-of-2 number of processing elements and a sufficient amount of available local PE memory , distributed FFTs of size p can be calculated on a ManArray processor since only hypercube connections are required. The hypercube of p or fewer nodes is a proper subset of the ManArray network. When p is a multiple of the number of processing elements, each PE emulates the operation of more than one virtual node. Therefore, any size of FFT problem can be handled using the above equation on any size of ManArray processor.

For direct execution, in other words, no emulation of virtual PEs, on a ManArray of size p, we need to provide a distributed FFT algorithm of equal size. For p=1, it is the sequential FFT. For p=2, the FFT of length 2 is the butterfly:

Y0=x0+w*X1, and

Y1=x0-w*X1

where X0 and Y0 reside in or must be saved in the local memory of PE0 and X1 and Y1 on PE1, respectively. The VLIWs in PE0 and PE1 in a 1x2 ManArray processor (p=2) that are required for the calculation of multiple FFTs of length 2 are shown in Fig. 9J which shows that two FFT results are produced every two cycles using four VLIWs.

Extending Complex Multiplication

It is noted that in the two-cycle complex multiplication hardware described in Figs. 6 and 7, the addition and subtraction blocks 623, 625, 723, and 725 operate in the second execution cycle. By including the MPYCX, MPYCXD2, MPYCXJ, and MPYCXJD2 instructions in the ManArray MAU, one of the execution units 131 of Fig. 1, the complex multiplication operations can be extended. The ManArray MAU also supports multiply accumulate operations (MACs) as shown in Figs. 11A and 12A for use in general digital signal processing (DSP) applications. A multiply accumulate instruction (MPYA) 1100 as shown in Fig. 11A, and a sum two product accumulate instruction (SUM2PA) 1200 as shown in Fig. 12A, are defined as follows.

In the MPYA instruction 1100 of Fig. 11A, the product of source registers *Rx* and *Ry* is added to target register *Rt*. The word multiply form of this instruction multiplies two 32-bit values producing a 64-bit result which is added to a 64-bit odd/even target register. The dual halfword form of MPYA instruction 1100 multiplies two pairs of 16-bit values producing two 32-bit results: one is added to the odd 32-bit word, the other is added to the even 32-bit word of the odd/even target register pair. Syntax and operation details 1110 are shown in Fig. 11B. In the SUM2PA instruction 1200 of Fig. 12A, the product of the high halfwords of source registers *Rx* and *Ry* is added to the product of the low halfwords of *Rx* and *Ry* and the result is added to target register *Rt* and then stored in *Rt*. Syntax and operation details 1210 are shown in Fig. 12B.

Both MPYA and SUMP2A generate the accumulate result in the second cycle of the two-cycle pipeline operation. By merging MPYCX, MPYCXD2, MPYCXJ, and MPYCXJD2 instructions with MPYA and SUMP2A instructions, the hardware supports the extension of the complex multiply operations with an accumulate operation. The mathematical operation is defined as: $Z_T = Z_R + X_R Y_R - X_I Y_I + i(Z_I + X_R Y_I + X_I Y_R)$, where $X = X_R + iX_I$, $Y = Y_R + iY_I$ and i is an imaginary number, or the square root of negative one, with $i^2 = -1$. This complex multiply accumulate is calculated in a variety of contexts, and it has been recognized that it will be highly advantageous to perform this calculation faster and more efficiently.

For this purpose, an MPYCXA instruction 1300 (Fig. 13A), an MPYCXAD2 instruction 1400 (Fig. 14A), an MPYCXJA instruction 1500 (Fig. 15A), and an MPYCXJAD2 instruction 1600 (Fig. 16A) define the special hardware instructions that handle the multiplication with accumulate for complex numbers. The MPYCXA instruction

1300, for multiplication of complex numbers with accumulate is shown in Fig. 13. Utilizing this instruction, the accumulated complex product of two source operands is rounded according to the rounding mode specified in the instruction and loaded into the target register. The complex numbers are organized in the source register such that halfword H1 contains the real component and halfword H0 contains the imaginary component. The MPYCXA instruction format is shown in Fig. 13A. The syntax and operation description 1310 is shown in Fig. 13B.

The MPYCXAD2 instruction 1400, for multiplication of complex numbers with accumulate, with the results divided by two is shown in Fig. 14A. Utilizing this instruction, the accumulated complex product of two source operands is divided by two, rounded according to the rounding mode specified in the instruction, and loaded into the target register. The complex numbers are organized in the source register such that halfword H1 contains the real component and halfword H0 contains the imaginary component. The MPYCXAD2 instruction format is shown in Fig. 14A. The syntax and operation description 1410 is shown in Fig. 14B.

The MPYCXJA instruction 1500, for multiplication of complex numbers with accumulate where the second argument is conjugated is shown in Fig. 15A. Utilizing this instruction, the accumulated complex product of the first source operand times the conjugate of the second source operand, is rounded according to the rounding mode specified in the instruction and loaded into the target register. The complex numbers are organized in the source register such that halfword H1 contains the real component and halfword H0 contains the imaginary component. The MPYCXJA instruction format is shown in Fig. 15A. The syntax and operation description 1510 is shown in Fig. 15B.

The MPYCXJAD2 instruction 1600, for multiplication of complex numbers with accumulate where the second argument is conjugated, with the results divided by two is shown in Fig. 16A. Utilizing this instruction, the accumulated complex product of the first source operand times the conjugate of the second operand, is divided by two, rounded according to the rounding mode specified in the instruction and loaded into the target register. The complex numbers are organized in the source register such that halfword H1 contains the real component and halfword H0 contains the imaginary component. The MPYCXJAD2 instruction format is shown in Fig. 16A. The syntax and operation description 1610 is shown in Fig. 16B.

All instructions of the above instructions 1100, 1200, 1300, 1400, 1500 and 1600 complete in two cycles and are pipeline-able. That is, another operation can start executing on the execution unit after the first cycle. All complex multiplication instructions 1300, 1400, 1500 and 1600 return a word containing the real and imaginary part of the complex product in half words H1 and H0 respectively.

To preserve maximum accuracy, and provide flexibility to programmers, the same four rounding modes specified previously for MPYCX, MPYCXD2, MPYCXJ, and MPYCXJD2 are used in the extended complex multiplication with accumulate.

Hardware 1700 and 1800 for implementing the multiply complex with accumulate instructions is shown in Fig. 17 and Fig. 18, respectively. These figures illustrate the high level view of the hardware 1700 and 1800 appropriate for these instructions. The important changes to note between Fig. 17 and Fig. 6 and between Fig. 18 and Fig. 7 are in the second stage of the pipeline where the two-input adder blocks 623, 625, 723, and 725 are replaced with three-input adder blocks 1723, 1725, 1823, and 1825. Further, two new half word source operands are used as inputs to the operation. The Rt.H1 1731 (1831) and Rt.H0 1733 (1833) values are properly aligned and selected by multiplexers 1735 (1835) and 1737 (1837) as inputs to the new adders 1723 (1823) and 1725 (1825). For the appropriate alignment, Rt.H1 is shifted right by 1-bit and Rt.H0 is shifted left by 15-bits. The add/subtract, add/sub blocks 1723 (1823) and 1725 (1825), operate on the input data and generate the outputs as shown. The add function and subtract function are selectively controlled functions allowing either addition or subtraction operations as specified by the instruction. The results are rounded and bits 30-15 of both 32-bit results are selected 1727 (1827) and stored in the appropriate half word of the target register 1729 (1829) in the CRF. It is noted that the multiplexers 1735 (1835) and 1737 (1837) select the zero input, indicated by the ground symbol, for the non-accumulate versions of the complex multiplication series of instructions.

While the present invention has been disclosed in the context of various aspects of presently preferred embodiments, it will be recognized that the invention may be suitably applied to other environments consistent with the claims which follow.

We claim:

1. An apparatus for the efficient processing of complex multiplication computations, the apparatus comprising:

at least one controller sequence processor (SP);

a memory for storing process control instructions;

a first multiply complex numbers instruction stored in the memory and operative to control the PEs to carry out a multiplication operation involving a pair of complex numbers; and

hardware for implementing the first multiply complex numbers instruction.

2. The apparatus of claim 1 further comprising a plurality of processing elements (PEs) interconnected with said SP and arranged in an N x N array interconnected in a manifold array interconnection network.

3. The apparatus of claim 1 wherein the first multiply complex instruction completes execution in 2 cycles.

4. The apparatus of claim 1 wherein the first multiply complex instruction is tightly pipelineable.

5. The apparatus of claim 1 wherein each complex number is stored as a word, each word comprising a first half word and a second half word, with a real component of each complex number being stored as the first half word and an imaginary component of each complex number being stored as the second half word.

6. The apparatus of claim 1 wherein the first multiply complex instruction includes a plurality of rounding modes, the rounding modes including:

rounding toward a nearest integer;

rounding toward zero;

rounding toward infinity; and

rounding toward negative infinity.

7. The apparatus of claim 1 wherein the first multiply complex numbers instruction

is one of the following group of instructions: a multiply complex numbers (MPYCX), a multiply complex numbers instruction (MPYCXJ) operative to carry out the multiplication of a pair of complex numbers where an argument is conjugated, a multiply complex numbers instruction (MPYCXD2) operative to carry out the multiplication of a pair of complex numbers with a result divided by two, and a multiply complex numbers instruction

(MPYCXJD2) operative to carry out the multiplication of a pair of complex numbers where an argument is conjugated with a result divided by two.

8. The apparatus of claim 1 further comprising a multiply accumulate unit including the memory for storing the first multiply complex numbers instruction.

9. The apparatus of claim 8 wherein the multiply accumulate unit operates in response to a multiply accumulate instruction (MPYA) to extend a multiplication operation with an accumulate operation.

10. The apparatus of claim 8 wherein the multiply accumulate unit operates in response to a sum two product accumulate instruction (SUM2PA) to extend two multiplication operations with an accumulate operation.

11. The apparatus of claim 9 wherein the multiply accumulate unit operates in response to a multiply complex with accumulate instruction (MPYCXA) to carry out the multiplication of a pair of complex numbers with accumulation of a third complex number.

12. The apparatus of claim 11 wherein the MPYCXA instruction completes execution in 2 cycles.

13. The apparatus of claim 12 wherein the MPYCXA instruction is tightly pipelineable.

14. The apparatus of claim 1 further comprising one or more of the following additional instructions (MPYCXA, MPYCXAD2, MPYCXJA or MPYCXJAD2) stored in the memory to carry out complex multiplication operations pipelined in 2 cycles.

15. A method for the computation of an FFT by a plurality of processing elements (PEs), the method comprising the steps of:

loading input data from a memory into each PE in a cyclic manner;

calculating a local FFT by each PE;

multiplying by the twiddle factors and calculating a FFT by the cluster of PEs; and

loading the FFTs into the memory.

16. A method for the computation of a distributed FFT by an N x N processing element (PE) array, the method comprising the steps of:

loading a complex number x and a corresponding twiddle factor w from a memory into each of the PEs;

calculating a first product by the multiplication of the complex numbers x and w;

transmitting the first product from each of the PEs to another PE in the N x N array;

receiving the first product and treating it as a second product in each of the PEs;

selectively adding or subtracting the first product and the second product to form a first result;

calculating a third product in selected PEs;

transmitting the first result or third product in selected PEs to another PE in the N x N array;

selectively adding or subtracting the received values to form a second result; and

storing the second results in the memory.

17. A method for efficient computation by a 2 x 2 processing element (PE) array interconnected in a manifold array interconnection network, the array comprising four PEs (PE0, PE1, PE2 and PE3), the method comprising the steps of:

loading a complex number x and a corresponding twiddle factor w from a memory into each of the four PEs, complex number x including subparts x0, x1, x2 and x3, twiddle factor w including subparts w0, w1, w2 and w3;

multiplying the complex numbers x and w, such that

PE0 multiplies x0 and w0 to produce a product0,

PE1 multiplies x1 and w1 to produce a product1,

PE2 multiplies x2 and w2 to produce a product2, and

PE3 multiplies x3 and w3 to produce a product3;

transmitting the product0, the product1, the product2 and the product3, such that

PE0 transmits the product0 to PE2,

PE1 transmits the product1 to PE3,

PE2 transmits the product2 to PE0, and

PE3 transmits the product3 to PE1; and

performing arithmetic logic operations, such that

PE0 adds the product0 and the product2 to produce a sum t0,

PE1 adds the product1 and the product3 to produce a sum t2,

PE2 subtracts the product2 from the product0 to produce a sum t1, and

PE3 subtracts the product3 from the product1 to produce a result which is multiplied by $-i$ to produce a sum t3.

18. The method of claim 17 further comprising the steps of:

transmitting the sums t0, t1, t2 and t3, such that

PE0 transmits t0 to PE1,

PE1 transmits t2 to PE0,

PE2 transmits t1 to PE3, and

PE3 transmits t3 to PE2;

performing the arithmetic logic operations, such that

PE0 adds t0 and t2 to produce a y0,

PE1 subtracts t2 from t0 to produce a y1,

PE2 adds t1 and t3 to produce a y2, and

PE3 subtracts t3 from t1 to produce a y3; and

storing y0, y1, y2 and y3 in a memory.

19. A special hardware instruction for handling the multiplication with accumulate for two complex numbers from a source register whereby utilizing said instruction and accumulated complex product of two source operands is rounded according to a rounding mode specified in the instruction and loaded into a target register with the complex numbers organized in the source such that a halfword (H1) contains the real component and a halfword (H0) contains the imaginary component.

20. The special hardware instruction of claim 19 wherein the accumulated complex product is divided by two before it is rounded.

21. An apparatus to efficiently fetch instructions including complex multiplication instructions and an accumulate form of multiplication instructions from a memory element and dispatch the fetched instruction to at least one of a plurality of multiply complex and multiply with accumulate execution units to carry out the instruction specified operation, the apparatus comprising:

a memory element;

means for fetching said instructions from the memory element;

a plurality of multiply complex and multiply with accumulate execution units; and

means to dispatch the fetched instruction to at least one of said plurality of execution units to carry out the instruction specified operation.

22. The apparatus of claim 21 further comprising:

an instruction register to hold a dispatched multiply complex instruction (MPYCX);

means to decode the MPYCX instruction and control the execution of the MPYCX instruction;

two source registers each holding a complex number as operand inputs to the multiply complex execution hardware;

four multiplication units to generate terms of the complex multiplication;

four pipeline registers to hold the multiplication results;

an add function which adds two of the multiplication results from the pipeline registers for the imaginary component of the result;

a subtract function which subtracts two of the multiplication results from the pipeline registers for the real component of the result;

a round and select unit to format the real and imaginary results; and

a result storage location for saving the final multiply complex result, whereby the apparatus is operative for the efficient processing of multiply complex computations.

23. The apparatus of claim 21 wherein the means for fetching said instructions is a sequence processor (SP) controller.

24. The apparatus of claim 22 wherein the round and select unit provides a shift right as a divide by 2 operation for a multiply complex divide by 2 instruction (MPYCXD2).

25. The apparatus of claim 21 further comprising:

an instruction register to hold a dispatched multiply complex instruction (MPYCXJ);

means to decode the MPYCXJ instruction and control the execution of the MPYCXJ instruction;

two source registers each holding a complex number as operand inputs to the multiply complex execution hardware;

four multiplication units to generate terms of the complex multiplication;

four pipeline registers to hold the multiplication results;

an add function which adds two of the multiplication results from the pipeline registers for the real component of the result;

a subtract function which subtracts two of the multiplication results from the pipeline registers for the imaginary component of the result;

a round and select unit to format the real and imaginary results; and

a result storage location for saving the final multiply complex conjugate result, whereby the apparatus is operative for the efficient processing of multiply complex conjugate computations.

26. The apparatus of claim 25 wherein the round and select unit provides a shift right as a divide by 2 operation for a multiply complex conjugate divide by 2 instruction (MPYCXJD2).

27. The apparatus of claim 21 further comprising:

an instruction register to hold the dispatched multiply accumulate instruction (MPYA);

means to decode the MPYA instruction and control the execution of the MPYA instruction;

two source registers each holding a source operand as inputs to the multiply accumulate execution hardware;

at least two multiplication units to generate two products of the multiplication;

at least two pipeline registers to hold the multiplication results;

at least two accumulate operand inputs to the second pipeline stage accumulate hardware;

at least two add functions which each adds the results from the pipeline registers with the third accumulate operand creating two multiply accumulate results;

a round and select unit to format the results if required by the MPYA instruction; and

a result storage location for saving the final multiply accumulate result, whereby the apparatus is operative for the efficient processing of multiply accumulate computations.

28. The apparatus of claim 21 further comprising:

an instruction register to hold a dispatched multiply accumulate instruction (SUM2PA);

means to decode the SUM2PA instruction and control the execution of the SUM2PA instruction;

at least two source registers each holding a source operand as inputs to the SUM2PA execution hardware;

at least two multiplication units to generate two products of the multiplication;

at least two pipeline registers to hold the multiplication results;

at least one accumulate operand input to the second pipeline stage accumulate hardware;

at least one add function which adds the results from the pipeline registers with the third accumulate operand creating a SUM2PA result;

a round and select unit to format the results if required by the SUM2PA instruction; and

a result storage location for saving the final result, whereby the apparatus is operative for the efficient processing of sum of 2 products accumulate computations

29. The apparatus of claim 21 further comprising:

an instruction register to hold the dispatched multiply complex accumulate instruction (MPYCXA);

means to decode the MPYCXA instruction and control the execution of the MPYCXA instruction;

two source registers each holding a complex number as operand inputs to the multiply complex accumulate execution hardware;

four multiplication units to generate terms of the complex multiplication;

four pipeline registers to hold the multiplication results;

at least two accumulate operand inputs to the second pipeline stage accumulate hardware;

an add function which adds two of the multiplication results from the pipeline registers and also adds one of the accumulate operand input for the imaginary component of the result;

a subtract function which subtracts two of the multiplication results from the pipeline registers and also adds the other accumulate operand input for the real component of the result;

a round and select unit to format the real and imaginary results; and

a result storage location for saving the final multiply complex accumulate result, whereby the apparatus is operative for the efficient processing of multiply complex accumulate computations.

30. The apparatus of claim 29 wherein the round and select unit provides a shift right as a divide by 2 operation for a multiply complex accumulate divide by 2 instruction (MPYCXAD2).

31. The apparatus of claim 21 further comprising:

an instruction register to hold the dispatched multiply complex conjugate accumulate instruction (MPYCXJA);

means to decode the MPYCXJA instruction and control the execution of the MPYCXJA instruction;

two source registers each holding a complex number as operand inputs to the multiply complex accumulate execution hardware;

four multiplication units to generate terms of the complex multiplication;

four pipeline registers to hold the multiplication results;

at least two accumulate operand inputs to the second pipeline stage accumulate hardware;

an add function which adds two of the multiplication results from the pipeline registers and also adds one of the accumulate operand input for the real component of the result;

a subtract function which subtracts two of the multiplication results from the pipeline registers and also adds the other accumulate operand input for the imaginary component of the result;

a round and select unit to format the real and imaginary results; and

a result storage location for saving the final multiply complex conjugate accumulate result, whereby the apparatus is operative for the efficient processing of multiply complex conjugate accumulate computations.

32. The apparatus of claim 31 wherein the round and select unit provides a shift right as a divide by 2 operation for a multiply complex conjugate accumulate divide by 2 instruction (MPYCXJAD2).

33. The apparatus of claim 21 wherein the complex multiplication instructions and accumulate form of multiplication instructions include MPYCX, MPYCXD2, MPYCXJ, MPYCXJD2, MPYCXA, MPYCXAD2, MPYCXJA, MPYCXJAD2 instructions, and all of these instructions complete execution in 2 cylces.

34. The apparatus of claim 21 wherein the complex multiplication instructions and accumulate form of multiplication instructions include MPYCX, MPYCXD2, MPYCXJ, MPYCXJD2, MPYCXA, MPYCXAD2, MPYCXJA, MPYCXJAD2 instructions, and all of these instructions are tightly pipelineable.

35. An apparatus for the efficient processing of an FFT, the apparatus comprising:

at least one controller sequence processor (SP);

a plurality of processing elements (PEs) arranged in an NxN array interconnected in a manifold (ManArray) interconnection network; and

a memory for storing instructions to be processed by the SP and by the array of PEs.

36. The apparatus of claim 22 wherein the add function and subtract function are selectively controlled functions allowing either addition or subtraction operations as specified by the instruction.

37. The apparatus of claim 25 wherein the add function and subtract function are selectively controlled functions allowing either addition or subtraction operations as specified by the instruction.

38. The apparatus of claim 29 wherein the add function and subtract function are selectively controlled functions allowing either addition or subtraction operations as specified by the instruction.

39. The apparatus of claim 31 wherein the add function and subtract function are selectively controlled functions allowing either addition or subtraction operations as specified by the instruction.

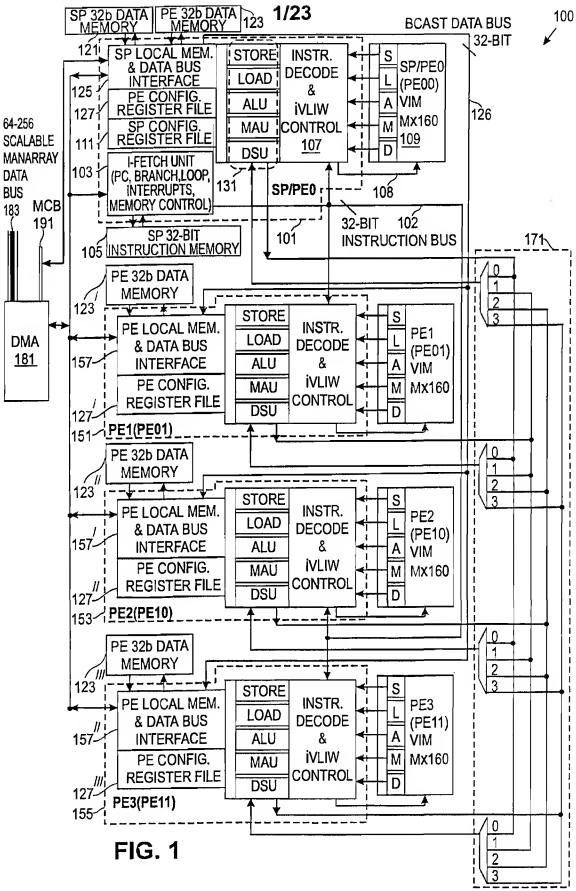

1/23
SP 32b DATA MEMORY
PE 32b DATA MEMORY
123
BCAST DATA BUS 32-BIT
100
121
SP LOCAL MEM. & DATA BUS INTERFACE
125
PE CONFIG. REGISTER FILE
127
SP CONFIG. REGISTER FILE
111
I-FETCH UNIT (PC, BRANCH,LOOP, INTERRUPTS, MEMORY CONTROL)
103
131
STORE
LOAD
ALU
MAU
DSU
INSTR. DECODE & iVLIW CONTROL 107
S
L
A
M
D
SP/PE0 (PE00) VIM MX160 109
126
64-256 SCALABLE MANARRAY DATA BUS 183
MCB 191
SP/PE0
108
101
32-BIT 102 INSTRUCTION BUS
171
SP 32-BIT INSTRUCTION MEMORY 105
PE 32b DATA MEMORY
123
DMA 181
PE LOCAL MEM. & DATA BUS INTERFACE
157
PE CONFIG. REGISTER FILE
127
PE1(PE01)
151
PE1 (PE01) VIM MX160
PE2(PE10)
153
PE2 (PE10) VIM MX160
PE3(PE11)
155
PE3 (PE11) VIM MX160
0
1
2
3

FIG. 1

200

MPYCX-MULTIPLY COMPLEX
ENCODING

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUP		S/P	UNIT		MAUopcode						Rt					Rx						Ry					CE3			0	RM	

FIG. 2A

210

Syntax/Operation

INSTRUCTION	OPERANDS	OPERATION	ACF
			Dual Halfwords
MPYCX.[SP]M.2SH	Rt, Rx, Ry RoundMode	Do operation below but do not affect ACFs	None
MPYCX.[CNVZ].[SP]M.2SH	Rt, Rx, Ry RoundMode	Rt.H1 ← round ((Rx.H1 * Ry.H1 - Rx.H0 * Ry.H0)/2^{15}) Rt.H0 ← round ((Rx.H1 * Ry.H0 + Rx.H0 * Ry.H1)/2^{15})	F1 F0
[TF].MPYCX.[SP]M.2SH	Rt, Rx, Ry RoundMode	Do operation only if T/F condition is satisfied in ACFs	None

ARITHMETIC SCALAR FLAGS AFFECTED (ON LEAST SIGNIFICANT OPERATION)
N=MSB OF RESULT
Z=1 IF ZERO RESULT IS GENERATED, 0 OTHERWISE
V=1 IF AN INTEGER OVERFLOW OCCURS ON EITHER OPERATION PRIOR TO THE DIVIDE, 0 OTHERWISE
C=NOT AFFECTED

CYCLES: 2

THE OPERAND RoundMode IS ONE OF THE FOLLOWING:

R=TRUNC FOR ROUNDING TOWARDS ZERO
R=CEIL FOR ROUNDING TOWARDS POSITIVE INFINITY
R=FLOOR FOR ROUNDING TOWARDS NEGATIVE INFINITY
R=ROUND FOR ROUNDING TO THE NEAREST INTEGER

FIG. 2B

300

MPYCXD2-MULTIPLY COMPLEX DIVIDE BY 2

ENCODING

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUP		S/P	UNIT		MAUopcode						Rt					Rx						Ry					CE3			0	RM	

FIG. 3A

310

Syntax/Operation

INSTRUCTION	OPERANDS	OPERATION	ACF
			Dual Halfwords
MPYCXD2.[SP]M.2SH	Rt, Rx, Ry RoundMode	Do operation below but do not affect ACFs	None
MPYCXD2[CNVZ].[SP]M.2SH	Rt, Rx, Ry RoundMode	Rt.H1 ← round ((Rx.H1 * Ry.H1 - Rx.H0 * Rx.H0)/2^{16}) Rt.H0 ← round ((Rx.H1 * Ry.H0 + Rx.H0 * Ry.H1)/2^{16})	F1 F0
[TF].MPYCXD2.[SP]M.2SH	Rt, Rx, Ry RoundMode	Do operation only if T/F condition is satisfied in ACFs	None

ARITHMETIC SCALAR FLAGS AFFECTED (ON LEAST SIGNIFICANT OPERATION)

N=MSB OF RESULT

Z=1 IF ZERO RESULT IS GENERATED, 0 OTHERWISE

V=1 IF AN INTEGER OVERFLOW OCCURS ON EITHER OPERATION PRIOR TO THE DIVIDE, 0 OTHERWISE

C=NOT AFFECTED

CYCLES: 2

THE OPERAND RoundMode IS ONE OF THE FOLLOWING:

R=TRUNC FOR ROUNDING TOWARDS ZERO
R=CEIL FOR ROUNDING TOWARDS POSITIVE INFINITY
R=FLOOR FOR ROUNDING TOWARDS NEGATIVE INFINITY
R=ROUND FOR ROUNDING TO THE NEAREST INTEGER

FIG. 3B

WO 00/22503
PCT/US99/23494

400

MPYCXJ-MULTIPLY COMPLEX CONJUGATE
ENCODING

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUP		S/P	UNIT		MAUopcode						Rt					Rx					Ry					CE3		0	RM			

FIG. 4A

410

Syntax/Operation

INSTRUCTION	OPERANDS	OPERATION	ACF
			Dual Halfwords
MPYCXJ.[SP]M.2SH	Rt, Rx, Ry RoundMode	Do operation below but do not affect ACFs	None
MPYCXJ[CNVZ].[SP]M.2SH	Rt, Rx, Ry RoundMode	Rt.H1 ← round ((Rx.H1 * Ry.H1 + Rx.H0 * Ry.H0)/2^{15}) Rt.H0 ← round ((Rx.H0 * Ry.H1 - Rx.H1 * Ry.H0)/2^{15})	F1 F0
[TF].MPYCXJ.[SP]M.2SH	Rt, Rx, Ry RoundMode	Do operation only if T/F condition is satisfied in ACFs	None

ARITHMETIC SCALAR FLAGS AFFECTED (ON LEAST SIGNIFICANT OPERATION)
N=MSB OF RESULT
Z=1 IF ZERO RESULT IS GENERATED, 0 OTHERWISE
V=1 IF AN INTEGER OVERFLOW OCCURS ON EITHER OPERATION, 0 OTHERWISE
C=NOT AFFECTED

CYCLES: 2

THE OPERAND RoundMode IS ONE OF THE FOLLOWING:

R=TRUNC FOR ROUNDING TOWARDS ZERO
R=CEIL FOR ROUNDING TOWARDS POSITIVE INFINITY
R=FLOOR FOR ROUNDING TOWARDS NEGATIVE INFINITY
R=ROUND FOR ROUNDING TO THE NEAREST INTEGER

FIG. 4B

PCT/US99/23494

500

MPYCXJD2-MULTIPLY COMPLEX CONJUGATE DIVIDE BY 2

ENCODING

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUP		S/P	UNIT		MAUopcode						Rt					Rx					Ry					CE3			0	RM		

FIG. 5A

510

Syntax/Operation

INSTRUCTION	OPERANDS	OPERATION	ACF
			Dual Halfwords
MPYCXJD2.[SP]M.2SH	Rt, Rx, Ry RoundMode	Do operation below but do not affect ACFs	None
MPYCXJD2[CNVZ].[SP]M.2SH	Rt, Rx, Ry RoundMode	Rt.H1 ← round ((Rx.H1 * Ry.H1 + Rx.H0 * Ry.H0)/2^{16}) Rt.H0 ← round ((Rx.H0 * Ry.H1 - Rx.H1 * Ry.H0)/2^{16})	F1 F0
[TF].MPYCXJD2.[SP]M.2SH	Rt, Rx, Ry RoundMode	Do operation only if T/F condition is satisfied in ACFs	None

ARITHMETIC SCALAR FLAGS AFFECTED (ON LEAST SIGNIFICANT OPERATION)
N=MSB OF RESULT
Z=1 IF ZERO RESULT IS GENERATED, 0 OTHERWISE
V=1 IF AN INTEGER OVERFLOW OCCURS ON EITHER OPERATION, 0 OTHERWISE
C=NOT AFFECTED

CYCLES: 2

THE OPERAND RoundMode IS ONE OF THE FOLLOWING:

R=TRUNC FOR ROUNDING TOWARDS ZERO
R=CEIL FOR ROUNDING TOWARDS POSITIVE INFINITY
R=FLOOR FOR ROUNDING TOWARDS NEGATIVE INFINITY
R=ROUND FOR ROUNDING TO THE NEAREST INTEGER

FIG. 5B

PCT/US99/23494

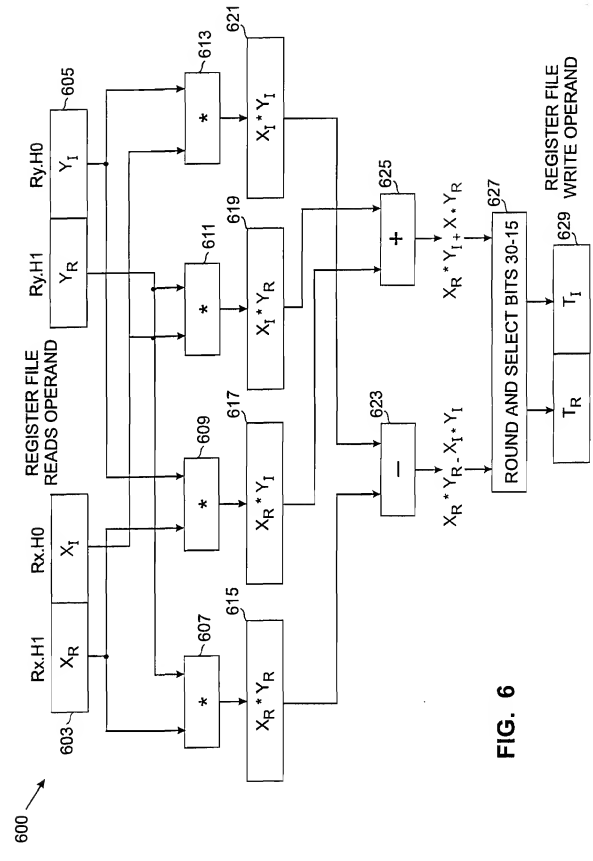

600
603
Rx.H1
X_R
Rx.H0
X_I
REGISTER FILE READS OPERAND
Ry.H1
Y_R
Ry.H0
Y_I
605
607
609
611
613
615
617
619
621
623
625
627
629
$X_R * Y_R$
$X_R * Y_I$
$X_I * Y_R$
$X_I * Y_I$
$X_R * Y_R - X_I * Y_I$
$X_R * Y_I + X_I * Y_R$
ROUND AND SELECT BITS 30-15
T_R
T_I
REGISTER FILE WRITE OPERAND

FIG. 6

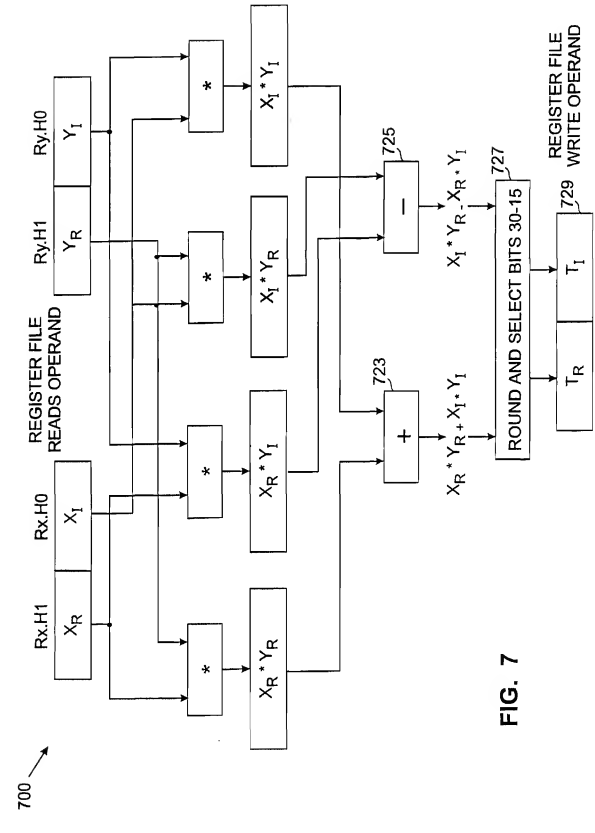

700
Rx.H1
Rx.H0
REGISTER FILE READS OPERAND
Ry.H1
Ry.H0
X_R
X_I
Y_R
Y_I
*
$X_R * Y_R$
$X_R * Y_I$
$X_I * Y_R$
$X_I * Y_I$
723
+
725
−
$X_R * Y_R + X_I * Y_I$
$X_I * Y_R - X_R * Y_I$
ROUND AND SELECT BITS 30-15
727
T_R
T_I
729
REGISTER FILE WRITE OPERAND

FIG. 7

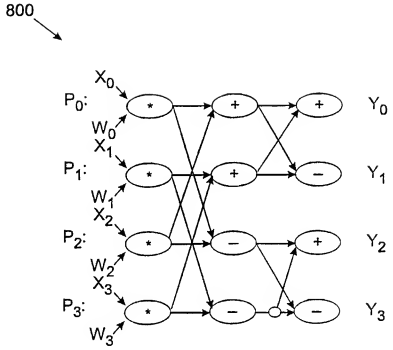


FIG. 8

INPUT OF THE DATA X AND ITS CORRESPONDING TWIDDLE FACTOR W:

LOAD	ALU	MAU	DSU	STORE
iii.p.w				
iii.p.w				

FIG. 9A

THE COMPLEX ARGUMENTS X AND W ARE MULTIPLIED:

LOAD	ALU	MAU	DSU	STORE
iii.p.w				
iii.p.w				
		mpycx.pm.2sh		

FIG. 9B

A COMMUNICATIONS OPERATIONS BETWEEN PEs:

LOAD	ALU	MAU	DSU	STORE
iii.p.w				
iii.p.w				
		mpycx.pm.2sh		
			pexchg.pd.w	

FIG. 9C

THE LOCAL AND RECEIVED QUANTITIES ARE ADDED OR SUBTRACTED (DEPENDING UPON THE PROCESSING ELEMENT.

LOAD	ALU	MAU	DSU	STORE
iii.p.w				
iii.p.w				
		mpycx.pm.2sh		
			pexchg.pd.w	
		sub.pa.2h		

FIG. 9D

THE RESULT IS MULTIPLIED BY -i ON PE3:

LOAD	ALU	MAU	DSU	STORE
lii.p.w				
lii.p.w				
		mpycx.pm.2sh		
			pexchg.pd.w	
	sub.pa.2h			
		mpycx.pm.2h		

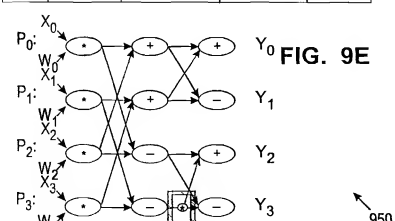


FIG. 9E

ANOTHER SET OF PE-TO-PE COMMUNICATIONS WHERE THE PREVIOUS PRODUCT IS EXCHANGED BETWEEN THE PES:

LOAD	ALU	MAU	DSU	STORE
lii.p.w				
lii.p.w				
		mpycx.pm.2sh		
			pexchg.pd.w	
	sub.pa.2h			
		mpycx.pm.2h		
			pexchg.pd.w	

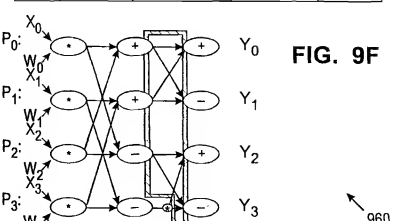


FIG. 9F

THE LOCAL AND RECEIVED QUANTITIES ARE ADDED OR SUBTRACTED (DEPENDING ON THE PROCESSING ELEMENT);

LOAD	ALU	MAU	DSU	STORE
lii.p.w				
lii.p.w				
		mpycx.pm.2sh		
			pexchg.pd.w	
	sub.pa.2h			
		mpycx.pm.2h		
			pexchg.pd.w	
	sub.pa.2h			

FIG. 9G

THE RESULTS ARE STORED TO LOCAL MEMORY:

LOAD	ALU	MAU	DSU	STORE
lii.p.w				
lii.p.w				
		mpycx.pm.2sh		
			pexchg.pd.w	
	sub.pa.2h			
		mpycx.pm.2h		
			pexchg.pd.w	
	sub.pa.2h			
				sli.p.w

FIG. 9H

VLIW	LOAD	ALU	MAU	DSU	STORE
	lii.p.w R0,a0=,1				
	lii.p.w R1,a1=,1				
	lii.p.w R0,a0=,1		mpycx.pm.2sh r2,r0,r1		
	lii.p.w R1,a1=,1				
	lii.p.w R0,a0=,1		mpycx.pm.2sh r2,r0,r1	pexchg.pd.w r3,r2,2x2_north	
	lii.p.w R1,a1=,1	sub.pa.2h r4,r2,r3			
	lii.p.w R0,a0=,1		mpycx.pm.2sh r2,r0,r1	pexchg.pd.w r3,r2,2x2_north	
	lii.p.w R1,a1=,1	sub.pa.2h r4,r2,r3	mpycx.pm.2h r5,r4,r8		
	lii.p.w R0,a0=,1		mpycx.pm.2sh r2,r0,r1	pexchg.pd.w r3,r2,2x2_north	
	lii.p.w R1,a1=,1	sub.pa.2h r4,r2,r3	mpycx.pm.2h r5,r4,r8	pexchg.pd.w r6,r5,2x2_east	
1	lii.p.w R0,a0=,1	sub.pa.2h r7,r5,r6	mpycx.pm.2sh r2,r0,r1	pexchg.pd.w r3,r2,2x2_north	
2	lii.p.w R1,a1=,1	sub.pa.2h r4,r2,r3	mpycx.pm.2h r5,r4,r8	pexchg.pd.w r6,r5,2x2_east	sii.p.w r7,a2+,1
1	lii.p.w R0,a0=,1	sub.pa.2h r7,r5,r6	mpycx.pm.2sh r2,r0,r1	pexchg.pd.w r3,r2,2x2_north	
2	lii.p.w R1,a1=,1	sub.pa.2h r4,r2,r3	mpycx.pm.2h r5,r4,r8	pexchg.pd.w r6,r5,2x2_east	sii.p.w r7,a2+,1

FIG. 9I

PE0

	LOAD	ALU	MAU	DSU	STORE
1		add.pa.4h r6,r4,r12		pexchg.pd.w r4,r2	
2	lii.p.d r2,a0+,1			pexchg.pd.w r5,r3	sii.p.d r6,a1+,1
3		add.pa.4h r6,r4,r2		pexchg.pd.w r4,r12	
4	lii.p.d r2,a0+,1			pexchg.pd.w r5,r13	sii.p.d r6,a1+,1

PE1

	LOAD	ALU	MAU	DSU	STORE
1	lii.p.d r8,a2+,1	sub.pa.4h r6,r4,r12	mpycx.pm.2sh r12,r0,r18	pexchg.pd.w r4,r2	
2	lii.p.d r0,a0+,1		mpycx.pm.2sh r13,r1,r19	pexchg.pd.w r5,r3	sii.p.d r6,a1+,1
3	lii.p.d r18,a2+,1	sub.pa.4h r6,r4,r2	mpycx.pm.2sh r2,r0,r8	pexchg.pd.w r4,r12	
4	lii.p.d r0,a0+,1		mpycx.pm.2sh r3,r1,r9	pexchg.pd.w r5,r13	sii.p.d r6,a1+,1

FIG. 9J

1010

$y = (I_m \otimes A)\, x$

Example: $y = (I_4 \otimes A)\, x_8$

$$\left(\begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix} \otimes \begin{bmatrix} a&b \\ c&d \end{bmatrix} \right) \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{bmatrix} = \begin{bmatrix} a&b&0&0&0&0&0&0 \\ c&d&0&0&0&0&0&0 \\ 0&0&a&b&0&0&0&0 \\ 0&0&c&d&0&0&0&0 \\ 0&0&0&0&a&b&0&0 \\ 0&0&0&0&c&d&0&0 \\ 0&0&0&0&0&0&a&b \\ 0&0&0&0&0&0&c&d \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{bmatrix} = \begin{bmatrix} ax_0 + bx_1 \\ cx_0 + dx_1 \\ ax_2 + bx_3 \\ cx_2 + dx_3 \\ ax_4 + bx_5 \\ cx_4 + dx_5 \\ ax_6 + bx_7 \\ cx_6 + dx_7 \end{bmatrix}$$

FIG. 10A

1020

$y = (A \otimes I_m)\, x$

Example: $y = (A \otimes I_4)\, x_8$

$$\left(\begin{bmatrix} a&b \\ c&d \end{bmatrix} \otimes \begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix} \right) \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{bmatrix} = \begin{bmatrix} a&0&0&0&b&0&0&0 \\ 0&a&0&0&0&b&0&0 \\ 0&0&a&0&0&0&b&0 \\ 0&0&0&a&0&0&0&b \\ c&0&0&0&d&0&0&0 \\ 0&c&0&0&0&d&0&0 \\ 0&0&c&0&0&0&d&0 \\ 0&0&0&c&0&0&0&d \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{bmatrix} = \begin{bmatrix} ax_0 + bx_4 \\ ax_1 + bx_5 \\ ax_2 + bx_6 \\ ax_3 + bx_7 \\ cx_0 + dx_4 \\ cx_1 + dx_5 \\ cx_2 + dx_6 \\ cx_3 + dx_7 \end{bmatrix}$$

FIG. 10B

1100

MPYA-MULTIPLY ACCUMULATE
ENCODING

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUP		S/P	UNIT		MAUopcode						RTE				0	Rx					Ry					0	CE2		MPack			

FIG. 11A

1110

Syntax/Operation

INSTRUCTION	OPERANDS	OPERATION	ACF
			Word
MPYA.[SP]M.1[SU]W	Rte, Rx, Ry	Do operation below but do not affect ACFs	None
MPYA[CNVZ].[SP]M.1[SU]W	Rte, Rx, Ry	RtollRte ← (Rx * Ry) + RtollRte	F0
[TF].MPYA.[SP]M.1[SU]W	Rte, Rx, Ry	Do operation only if T/F condition is satisfied in ACFs	None
			Dual Halfwords
MPYA.[SP]M.2[SU]H	Rte, Rx, Ry	Do operation below but do not affect ACFs	None
MPYA[CNVZ].[SP]M.1[SU]H	Rte, Rx, Ry	Rto ← (Rx.H1 * Ry.H1) + Rto	F1
		Rte ← (Rx.H0 * Ry.H0) + Rte	F0
[TF].MPYA.[SP]M.2[SU]H	Rte, Rx, Ry	Do operation only if T/F condition is satisfied in ACFs	None

ARITHMETIC FLAGS AFFECTED
N=MSB OF RESULT
Z=1 IF RESULT IS ZERO, 0 OTHERWISE
V=1 IF AN OVERFLOW OCCURS ON THE ADDITION, 0 OTHERWISE
C=1 IF AN CARRY OCCURS ON THE ADDITION, 0 OTHERWISE

FIG. 11B

CYCLES: 2

1200

SUM2PA - SUM OF 2 PRODUCTS ACCUMULATE

ENCODING

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUP		S/P	UNIT		MAUopcode						Rt					Rx					Ry				CE3	SumpExt						
											Rte				0	Rxe				0	Rye			0								

FIG. 12A

1210

Syntax/Operation

INSTRUCTION	OPERANDS	OPERATION	ACF
			Dual Halfwords
SUM2PA.[SP]M.2[SU]H	Rt, Rx, Ry	Do operation below but do not affect ACFs	None
SUM2PA[CNVZ].[SP]M.2[SU]H	Rt, Rx, Ry	Rt ← Rt + (Rx.H1 * Ry.H1) + (Rx.H0 * Ry.H0)	F0
[TF].SUM2PA.[SP]M.2[SU]H	Rt, Rx, Ry	Do operation only if T/F condition is satisfied in ACFs	None
			Quad Halfwords
SUM2PA.[SP]M.4[SU]H	Rte, Rxe, Rye	Do operation below but do not affect ACFs	None
SUM2PA[CNVZ].[SP]M.4[SU]H	Rte, Rxe, Rye	Rto ← Rto+ (Rxo.H1 * Ryo.H1) + (Rxo.H0 * Ryo.H0)	F1
		Rte ← Rte+ (Rxe.H1 * Rye.H1) + (Rxe.H0 * Rye.H0)	F0
[TF].SUM2PA.[SP]M.4[SU]H	Rte, Rxe, Rye	Do operation only if T/F condition is satisfied in ACFs	None

ARITHMETIC SCALAR FLAGS AFFECTED (on least significant operation)

N=MSB OF RESULT

Z=1 IF RESULT IS ZERO, 0 OTHERWISE

V=1 IF AN OVERFLOW OCCURS ON THE ADD WITH Rt, 0 OTHERWISE

C=1 IF AN CARRY OCCURS ON THE ADD WITH Rt,, 0 OTHERWISE

FIG. 12B

CYCLES: 2

1300

MPYCXA-MULTIPLY COMPLEX ACCUMULATE
ENCODING

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUP		S/P	UNIT		MAUopcode						Rt					Rx						Ry					CE3			0	RM	

FIG. 13A

1310

Syntax/Operation

INSTRUCTION	OPERANDS	OPERATION	ACF
			Dual Halfwords
MPYCXA.[SP]M.2SH	Rt, Rx, Ry RoundMode	Do operation below but do not affect ACFs	None
MPYCXA[CNVZ].[SP]M.2SH	Rt, Rx, Ry	Rt.H1 ← round ((Rt.H1+ Rx.H1 * Ry.H1 - Rx.H0 * Ry.H0)/2^{15})	F1
	RoundMode	Rt.H0 ← round ((Rt.H0+ Rx.H1 * Ry.H0 + Rx.H0 * Ry.H1)/2^{15})	F0
[TF].MPYCXA.[SP]M.2SH	Rt, Rx, Ry RoundMode	Do operation only if T/F condition is satisfied in ACFs	None

ARITHMETIC SCALAR FLAGS AFFECTED (ON LEAST SIGNIFICANT OPERATION)
N=MSB OF RESULT
Z=1 IF ZERO RESULT IS GENERATED, 0 OTHERWISE
V=1 IF AN INTEGER OVERFLOW OCCURS ON EITHER OPERATION PRIOR TO THE DIVIDE, 0 OTHERWISE
C=NOT AFFECTED

THE OPERAND RoundMode IS ONE OF THE FOLLOWING:
R=TRUNC FOR ROUNDING TOWARDS ZERO
R=CEIL FOR ROUNDING TOWARDS POSITIVE INFINITY
R=FLOOR FOR ROUNDING TOWARDS NEGATIVE INFINITY
R=ROUND FOR ROUNDING TO THE NEAREST INTEGER

CYCLES: 2

FIG. 13B

PCT/US99/23494

1400

MPYCXAD2-MULTIPLY COMPLEX ACCUMULATE DIVIDE BY 2

ENCODING

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUP		S/P	UNIT		MAUopcode						Rt					Rx						Ry					CE3			0	RM	

FIG. 14A

1410

Syntax/Operation

INSTRUCTION	OPERANDS	OPERATION	ACF
			Dual Halfwords
MPYCXAD2.[SP]M.2SH	Rt, Rx, Ry RoundMode	Do operation below but do not affect ACFs	None
MPYCXAD2[CNVZ].[SP]M.2SH	Rt, Rx, Ry RoundMode	Rt.H1 ← round ((Rt.H1+ Rx.H1 * Ry.H1 - Rx.H0 * Ry.H0)/2^{16}) Rt.H0 ← round ((Rt.H0+ Rx.H1 * Ry.H0 + Rx.H0 * Ry.H1)/2^{16})	F1 F0
[TF].MPYCXAD2.[SP]M.2SH	Rt, Rx, Ry RoundMode	Do operation only if T/F condition is satisfied in ACFs	None

ARITHMETIC SCALAR FLAGS AFFECTED (ON LEAST SIGNIFICANT OPERATION)

N=MSB OF RESULT
Z=1 IF ZERO RESULT IS GENERATED, 0 OTHERWISE
V=1 IF AN INTEGER OVERFLOW OCCURS ON EITHER OPERATION PRIOR TO THE DIVIDE, 0 OTHERWISE
C=NOT AFFECTED

THE OPERAND RoundMode IS ONE OF THE FOLLOWING:

R=TRUNC FOR ROUNDING TOWARDS ZERO
R=CEIL FOR ROUNDING TOWARDS POSITIVE INFINITY
R=FLOOR FOR ROUNDING TOWARDS NEGATIVE INFINITY
R=ROUND FOR ROUNDING TO THE NEAREST INTEGER

CYCLES: 2

FIG. 14B

1500

MPYCXJ-MULTIPLY COMPLEX CONJUGATE ACCUMULATE

ENCODING

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUP			S/P	UNIT	MAUopcode						Rt					Rx						Ry					CE3		0	RM		

FIG. 15A

1510

Syntax/Operation

INSTRUCTION	OPERANDS	OPERATION	ACF
			Dual Halfwords
MPYCXJA.[SP]M.2SH	Rt, Rx, Ry RoundMode	Do operation below but do not affect ACFs	None
MPYCXJA[CNVZ].[SP]M.2SH	Rt, Rx, Ry RoundMode	Rt.H1 ← round ((Rt.H1+ Rx.H1 * Ry.H1 + Rx.H0 * Ry.H0)/2^{15}) Rt.H0 ← round ((Rt.H0+ Rx.H0 * Ry.H1 - Rx.H1 * Ry.H0/2^{15})	F1 F0
[TF].MPYCXJA.[SP]M.2SH	Rt, Rx, Ry RoundMode	Do operation only if T/F condition is satisfied in ACFs	None

ARITHMETIC SCALAR FLAGS AFFECTED (ON LEAST SIGNIFICANT OPERATION)

N=MSB OF RESULT
Z=1 IF ZERO RESULT IS GENERATED, 0 OTHERWISE
V=1 IF AN INTEGER OVERFLOW OCCURS ON EITHER, 0 OTHERWISE
C=NOT AFFECTED

CYCLES: 2

THE OPERAND RoundMode IS ONE OF THE FOLLOWING:

R=TRUNC FOR ROUNDING TOWARDS ZERO
R=CEIL FOR ROUNDING TOWARDS POSITIVE INFINITY
R=FLOOR FOR ROUNDING TOWARDS NEGATIVE INFINITY
R=ROUND FOR ROUNDING TO THE NEAREST INTEGER

FIG. 15B

1600

MPYCXJD2-MULTIPLY COMPLEX CONJUGATE ACCUMULATE DIVIDE BY 2 ENCODING

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUP		S/P	UNIT		MAUopcode						Rt					Rx					Ry					CE3			0	RM		

FIG. 16A

1610

Syntax/Operation

INSTRUCTION	OPERANDS	OPERATION	ACF
			Dual Halfwords
MPYCXJAD2.[SP]M.2SH	Rt, Rx, Ry RoundMode	Do operation below but do not affect ACFs	None
MPYCXJAD2[CNVZ].[SP]M.2SH	Rt, Rx, Ry RoundMode	Rt.H1 ← round ((Rt.H1+ Rx.H1 * Ry.H1 + Rx.H0 * Ry.H0)/2^{16}) Rt.H0 ← round ((Rt.H0+ Rx.H0 * Ry.H1 - Rx.H1 * Ry.H0/2^{16})	F1 F0
[TF].MPYCXJAD2.[SP]M.2SH	Rt, Rx, Ry RoundMode	Do operation only if T/F condition is satisfied in ACFs	None

ARITHMETIC SCALAR FLAGS AFFECTED (ON LEAST SIGNIFICANT OPERATION)
N=MSB OF RESULT
Z=1 IF ZERO RESULT IS GENERATED, 0 OTHERWISE
V=1 IF AN INTEGER OVERFLOW OCCURS ON EITHER OPERATION, 0 OTHERWISE
C=NOT AFFECTED

CYCLES: 2

THE OPERAND RoundMode IS ONE OF THE FOLLOWING:

R=TRUNC FOR ROUNDING TOWARDS ZERO
R=CEIL FOR ROUNDING TOWARDS POSITIVE INFINITY
R=FLOOR FOR ROUNDING TOWARDS NEGATIVE INFINITY
R=ROUND FOR ROUNDING TO THE NEAREST INTEGER

FIG. 16B

WO 00/22503
PCT/US99/23494

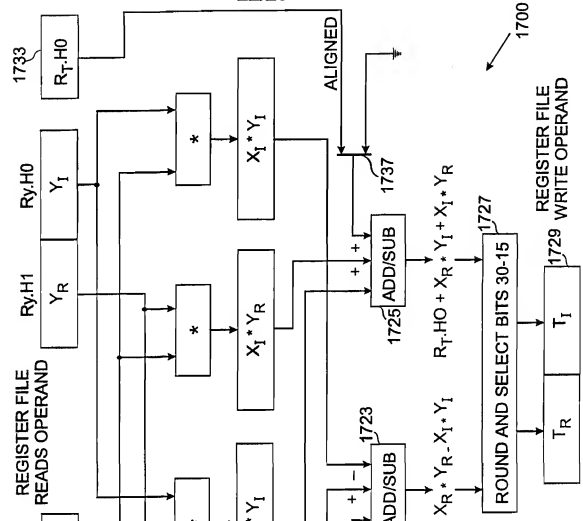

1731
R_T.H1
Rx.H1
X_R
Rx.H0
X_I
REGISTER FILE READS OPERAND
Ry.H1
Y_R
Ry.H0
Y_I
1733
R_T.H0
ALIGNED
$X_R * Y_R$
$X_R * Y_I$
$X_I * Y_R$
$X_I * Y_I$
1735
1723
ADD/SUB
1725
ADD/SUB
1737
R_T.H1+ $X_R * Y_R$ - $X_I * Y_I$
R_T.HO + $X_R * Y_I$ + $X_I * Y_R$
1727
ROUND AND SELECT BITS 30-15
T_R
T_I
1729
REGISTER FILE WRITE OPERAND
1700

FIG. 17

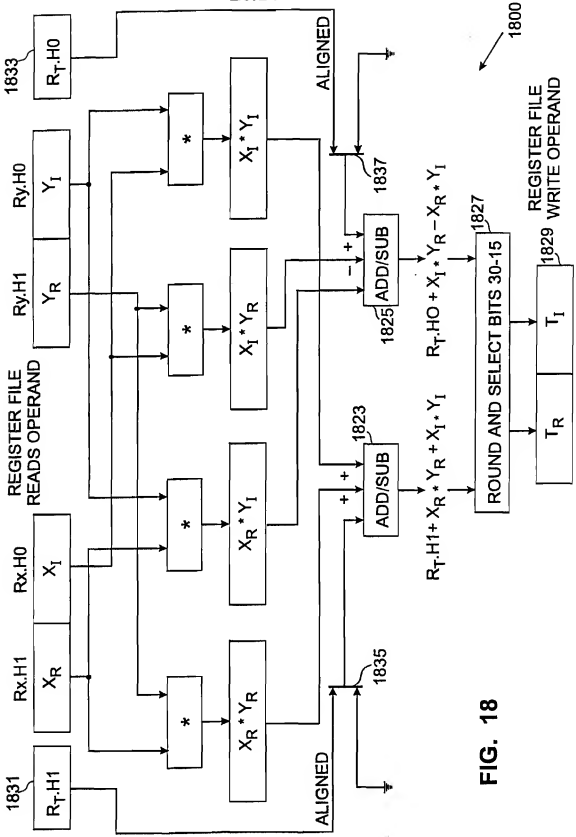

1831
R_T.H1
Rx.H1
X_R
Rx.H0
X_I
REGISTER FILE READS OPERAND
Ry.H1
Y_R
Ry.H0
Y_I
1833
R_T.H0
ALIGNED
1835
$X_R * Y_R$
$X_R * Y_I$
$X_I * Y_R$
$X_I * Y_I$
1823
ADD/SUB
1825
ADD/SUB
1837
R_T.H1+ $X_R * Y_R + X_I * Y_I$
R_T.H0 + $X_I * Y_R - X_R * Y_I$
1827
ROUND AND SELECT BITS 30-15
1829
T_R
T_I
REGISTER FILE WRITE OPERAND
1800

FIG. 18

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/23494

A. CLASSIFICATION OF SUBJECT MATTER
IPC(6) :G06F 7/50, 7/52, 15/16, 15/80, 17/10, 17/14
US CL :712/11, 16, 205; 708/232, 404, 524, 622
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 712/11, 16, 205; 708/232, 404, 524, 622

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

STN/CAS

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 4,709,327 A (HILLIS et al) 24 November 1987, cols. 1-46.	1-14, 19-34, 36-39
Y	US 5,367,692 A (EDELMAN) 22 November 1994, cols. 1-22.	1-14, 19-20
Y	US 5,659,785 A (PECHANEK et al) 19 August 1997, cols. 1-18.	15, 16, 35
Y	US 5,649,135 A (PECHANEK et al) 15 July 1997, cols. 1-20.	15, 16, 35
Y	US 5,682,491 A (PECHANEK et al) 28 October 1997, cols. 1-18.	15, 16, 35
Y	US 5,428,754 A (BALDWIN) 27 June 1995, cols. 1-122.	17, 18

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance		
"E"	earlier document published on or after the international filing date	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed	"&"	document member of the same patent family

Date of the actual completion of the international search 06 DECEMBER 1999	Date of mailing of the international search report 07 FEB 2000
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer JOHN FOLLANSBEE Telephone No. (703) 305-8498

Form PCT/ISA/210 (second sheet)(July 1992) ★

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/23494

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5,179,530 A (GENUSOV et al) 12 January 1993, cols. 1-20.	17, 18
Y	US 5,798,753 A (ZHOU et al) 25 August 1998, cols. 1-18.	21-34, 36-39

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/23494

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This international report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

Please See Extra Sheet.

1. ☒ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☐ No protest accompanied the payment of additional search fees.

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/23494

BOX II. OBSERVATIONS WHERE UNITY OF INVENTION WAS LACKING
This ISA found multiple inventions as follows:

This application contains the following inventions or groups of inventions which are not so linked as to form a single inventive concept under PCT Rule 13.1. In order for all inventions to be searched, the appropriate additional search fees must be paid.

Group I, claims 1-14 and 19-20, drawn to an array processor for performing multiplications on complex numbers;

Group II, claim 15, drawn to an array processor for computing an FFT using twiddle factors;

Group III, claim 16, drawn to an array processor for computing a distributed FFT using twiddle factors, multiplication, adding or substracting;

Group IV, claims 17-18, drawn to an array processor for multiplying specific subparts of complex numbers and twiddle factors;

Group V, claims 21-34, 36-39, drawn to fetching complex multiplication instructions and dispatching of the instructions; and

Group VI, claim 35, drawn to an array processor for performing efficient processing of an FFT.

The inventions listed as Groups I-VI do not relate to a single inventive concept under PCT Rule 13.1 because, under PCT Rule 13.2, they lack the same or corresponding special technical features for the following reasons; Group I lacks the particulars of: the calculation of an FFT using twiddle factors of Group II; the calculation of a distributed FFT using twiddle factors, multiplication, adding or subtracting; the computation of multiplying specific subparts of complex numbers and twiddle factors; the fetching and dispatching of complex multiplication instructions; and for performing efficient processing of an FFT. As per Groups II to Group VI, they lack the same or corresponding special technical features with respect to each other for similar reasons as stated above.